Reusable Components

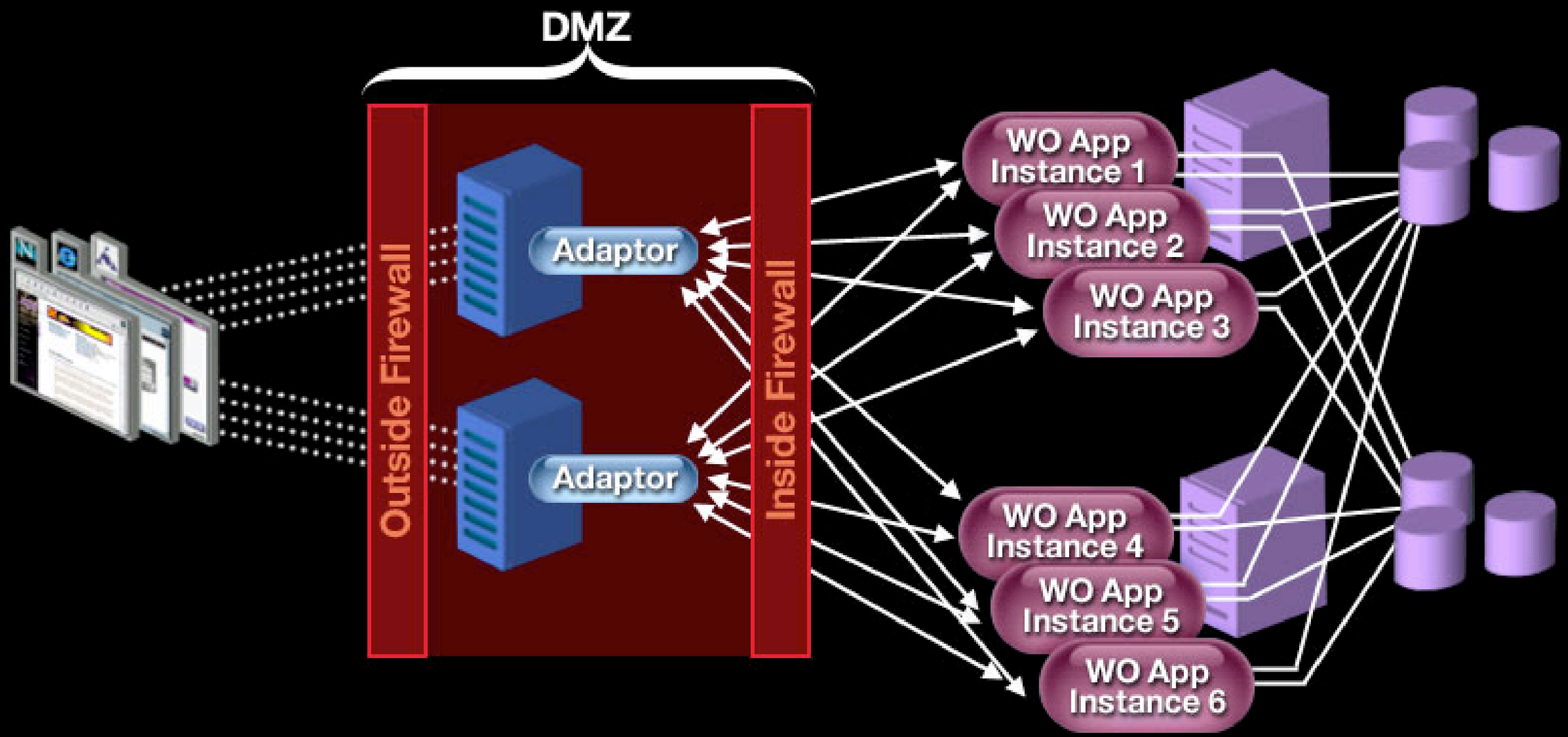
Session 614

WebObjects: Reusable Components

Fitzpatrick, Leung, Neumann

Reusable Slides

Web App Architectural Overview



Browsers

Multiple HTTP Servers

Multiple WO App Servers

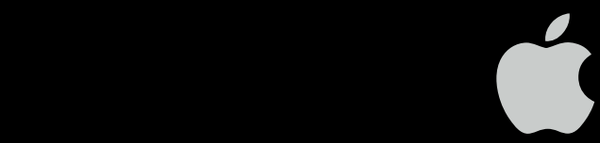
Multiple Databases

WWDC2002

Reusable Jokes

What's the Longest Method Name In WebObjects?

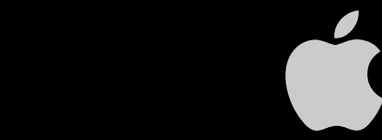
- addObjectToBothSidesOfRelationshipWithKey()
- databaseContextWillRunLoginPanelToOpenDatabaseChannel()
- adaptorChannelShouldConstructStoredProcedureReturnValues()
- objectStoreCoordinatorShouldWaitForOtherObjectStoreCoordinatorsAndRunOutForBeerAndChipsLeavingKidsAtHomeWithTheAdaptorInContext()
- statementsToImplementForeignKeyConstraints...



- ...OnEntityGroupsWithAdaptorChannelObjectStoreCoordinatorDidEvaluateExpressionWithFetchSpecificationAndKeyThenCraigFederighiSentAnEmailAboutNextGenerationJDOCompatabilityOfStatementRegardingMattsFoosballAcumenBeingLessDramaticThanRonLueSangWhoEveryoneKnowsPracticesAtHomeButTellsHisManagerHeIsTakingValuesFromAppendToResponseAndTariqAzizSaysLookAtMeIAmNotSurprisedAndAndreasSaidThatsMyJokeYouCantTakeThatJokeAndStanJirmanJustDroveOverMyTiBookFiringAFaultForAdaptorChannelInContext()

What Is the Most Popular WebObjects Movie This Summer?

- NSMatrix Reloaded
- XCode 2: Code United
- NSDebugRats
- Terminate Application III: Rise of the Virtual Machines
- Finding EO



What's the Best Name for the Release Notes to Get People to Read Them?

- "Don't Read Me"
- "Nothing to see Here"
- "EOEditingContext.java"
- "SJ-Macworld-2004.key"
- "G6 Specifications"

WebObjects: Reusable Components

Commitment

- If EOF is about relationships, reusable components are about bindings and commitment
- One should strive for loose bindings and low commitment
- What do we mean by commitment?
  - Frameworks to include
  - Interfaces that need to be adhered to
  - Only works with EOEnterpriseObjects

Agenda

- Introduction
- Bindings
- Component Content
- Streaming HTTP requests
- Tips and Tricks
- Dr. Evil
- Examples

Introduction

- Reusable components are WebObjects' GUI widgets
- Subclass of WOComponent; there is no distinction between reusable components and pages
  - Edit them in WebObjectsBuilder
- We talk about "child" components being used or embedded in "parent" components
  - Can be nested arbitrarily deep, though there is a performance cost to be paid
- Use them just like WOString or WOHyperlink

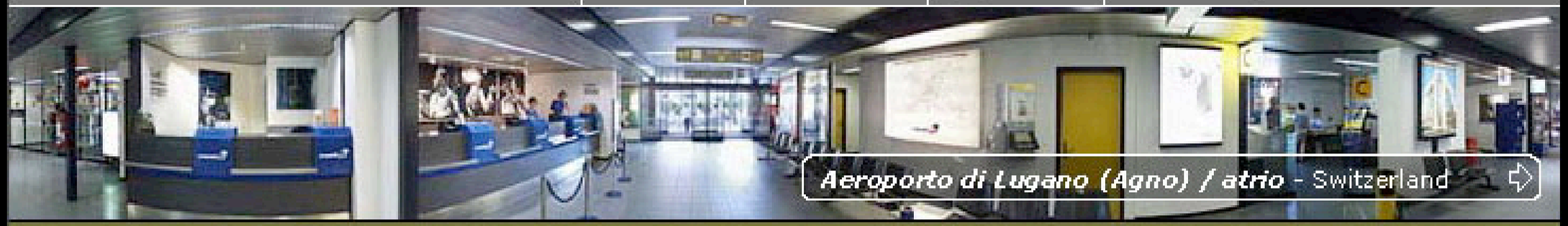
Design Patterns

- Templates
- Minicomponents

Travel the

MAPS | CHANNELS | MAG | SEARCH

ABOUT US
CONTACT US
HELP



CITY LOCATOR

Choose a Country | GO | Choose a City

Join the community

QUICK SEARCH

GO

Welcome to map .com

DESTINATION

SEARCH

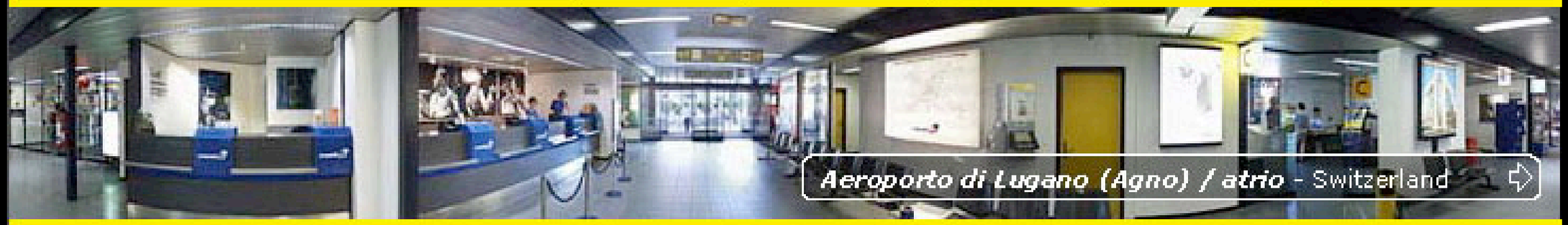
■ **Navigate the world with Maps**, and zoom in to the destination of your choice. A few clicks will take you from a map right to where you want to be. From continent to country, region to city, street to monument

your world, your way

Travel the

MAPS | CHANNELS | MAG | SEARCH

ABOUT US
CONTACT US
HELP



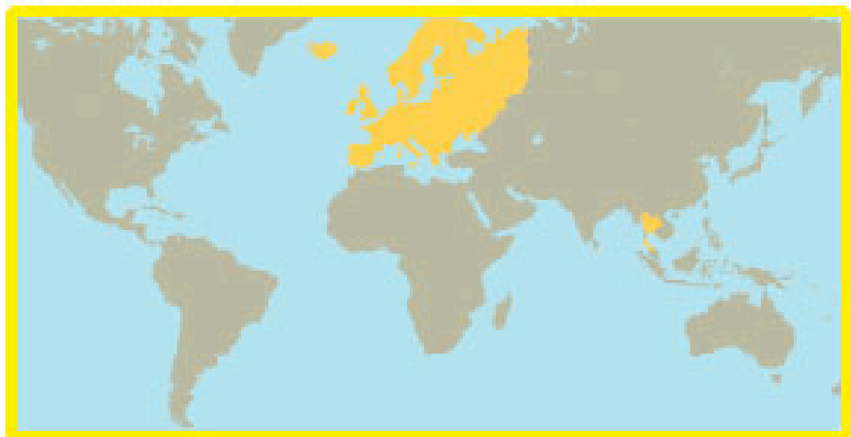
CITY LOCATOR

Choose a Country | GO | Choose a City

Join the community

QUICK SEARCH

GO

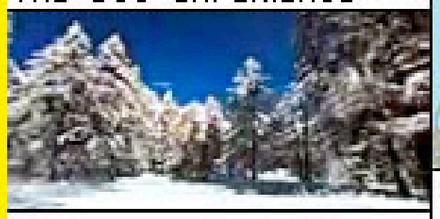


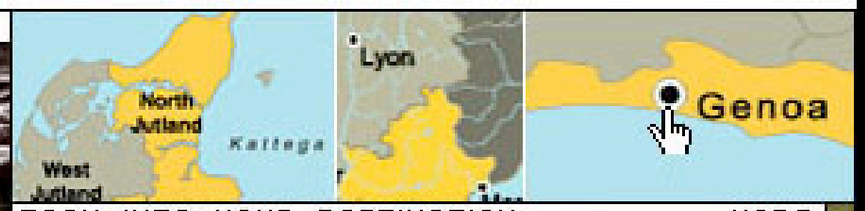
DESTINATION

SEARCH

Welcome to maps.[redacted].com

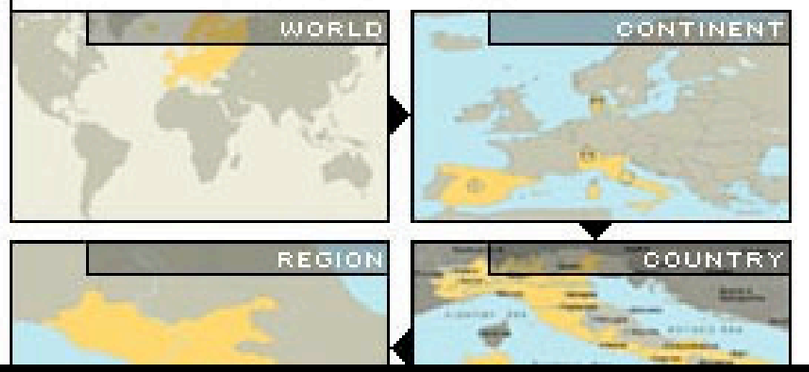
THE 360 EXPERIENCE





ZOOM INTO YOUR DESTINATION

MAPS

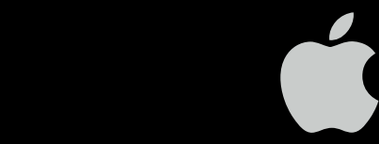


■ **Navigate the world with Maps**, and zoom in to the destination of your choice. A few clicks will take you from a map right to where you want to be. From continent to country, region to city, street to monument

Bindings

Bindings

- Mechanism through which a reusable component communicates with its parent
  - Can be considered “arguments” to reusable components
- Bindings are “pulled” into the child, or “pushed” into the parent
- The child controls pulling and pushing of bindings, not the parent
- Fill out the API for your component’s .wo—it’s your best documentation



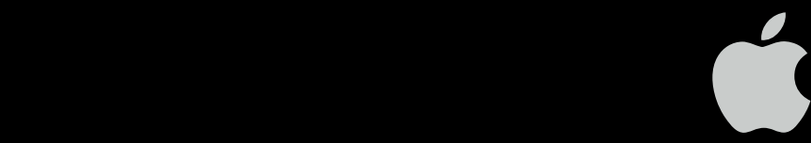
Many Ways of Handling Bindings

- Auto-synchronization
- Manual synchronization using valueForBinding(), setValueForBinding(), hasBinding()
- ‘^’ carat notation
- Accessor methods
- Accessor methods with cache
- Default values for bindings
- Page level bindings
- And more...

Auto-Synchronization

What is it?

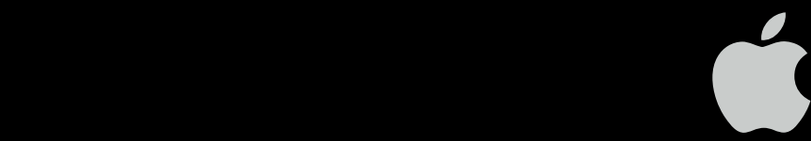
- WebObjects' mechanism for passing binding values at the "appropriate" times
- Bindings are pulled and pushed at the beginning and end of takeValuesFromRequest(), invokeActionForRequest(), and appendToResponse()
- WebObjects' default
  - Does not preclude manual synchronization
  - Override synchronizesVariablesWithBindings() in your reusable component to go fully manual



Auto-Synchronization

Why Is It Undesireable?

- It's magic
- It happens more often than necessary
  - Bad for performance
  - Have to worry about side effects in your accessor methods
- The order in which bindings are pushed and pulled is indeterminate
  - What if you need to compute something based on more than one binding value?



Accessor Methods With Cache

- Cache binding values to improve performance
- Flush caches at the beginning of appendToResponse()

Accessor Methods With Cache

ReusableComponent.java

```
private String binding;

public String binding() {
    if ( binding == null ) {
        binding = valueForBinding(“binding”);
    }
    return binding;
}

public void setBinding(String value) {
    binding = value;
    setValueForBinding(binding, “binding”);
}

public void appendToResponse(WOResponse response, WOContext context) {
    binding = null;
    super.appendToResponse(response, context);
}
```

Default Values for Bindings

- Simply check again to see if binding value is null
- setBinding() automatically pushes value back up to the parent

ReusableComponent.java

```
private String binding;

public String binding() {
    if ( binding == null ) {
        binding = valueForBinding(“binding”);
        if ( binding == null ) {
            setBinding(DEFAULT_BINDING_VALUE);
        }
    }
    return binding;
}
```

Page Level Bindings

- Convenient way to propagate values like “editable” to all components without explicitly wiring them up
- Allow override by explicitly providing a binding
- Similarly have session and application level bindings

Page Level Bindings

ReusableComponent.java

```
public Boolean editable() {
    if ( hasBinding(“editable”) ) {
        return valueForBinding(“editable”);
    }
    try {
        return root().valueForKey(“editable”);
    } catch ( NSKeyValueCoding.UnknownKeyException ) {
        return null;
    }
}
```

Component Content

ComponentContent vs. SwitchComponent

- Both allow Reusable Components with arbitrary, dynamic content
- SwitchComponent
  - Requires existence of an extra Reusable Component
  - Knowledge of expected possible bindings on those Reusable Components
- ComponentContent
  - Lets the parent define any template on the spot
  - No constraint on how the dynamic area is defined

Why SwitchComponent?

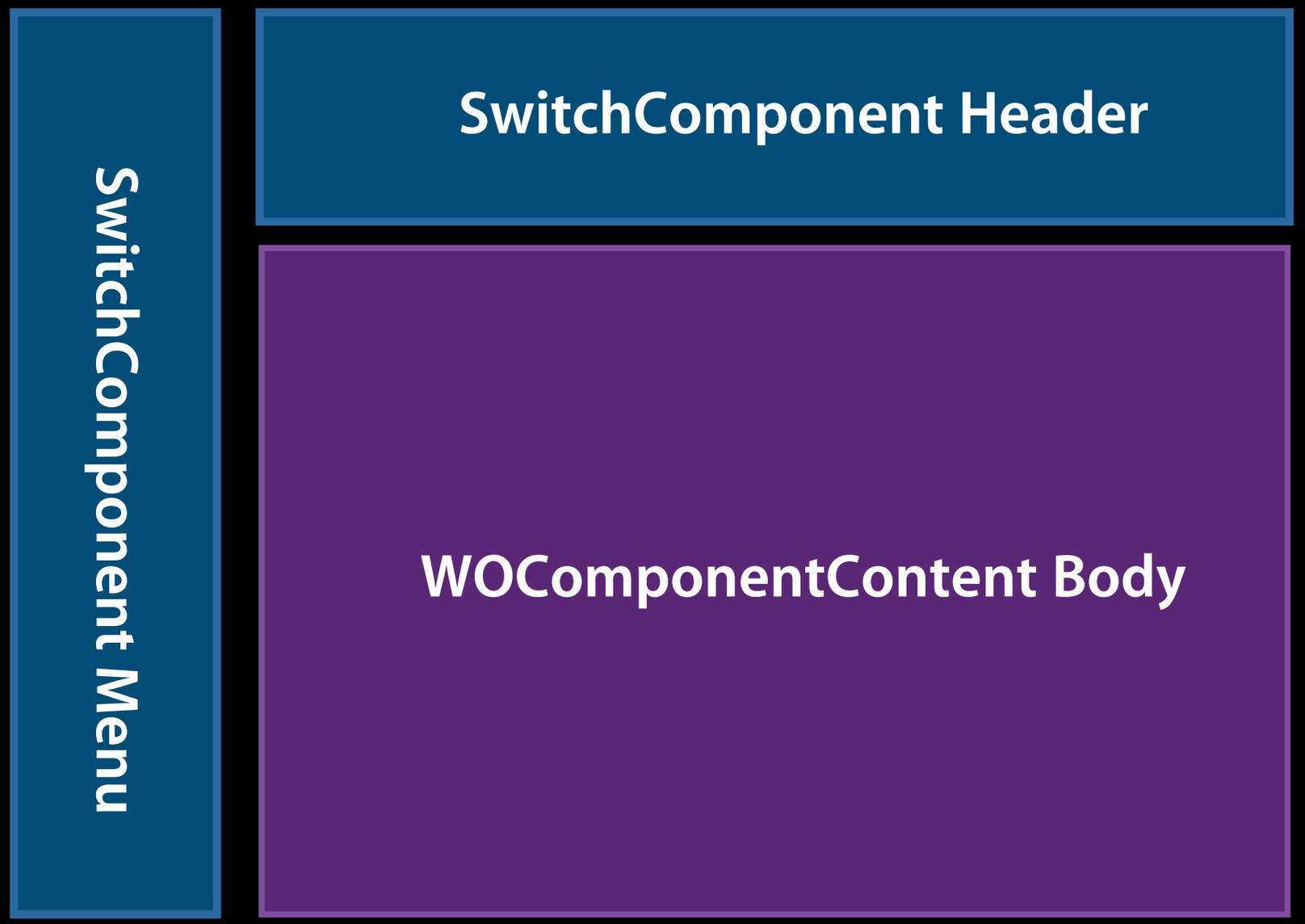
- Your reusable component wanted to express itself in certain limited but extensible ways
- You didn’t want the parent to have to provide anything
- You need more than one dynamic region

Multiple Component Content

- You often wish your RC could use more than one WOComponentContent
- Multiple Component Content remove this constraint
- Implemented with two WODynamicElements provided with source in our Example project
  - RCComponentContent
  - RCTemplate

Multiple Custom Contents

Without Using MultipleComponentContent



Multiple Custom Contents

With MultipleComponentContent

MultipleComponentContent Menu

MultipleComponentContent Header

MultipleComponentContent Body

MultipleComponentContent

Reusable Component

ReusableComponent.html

```
<TABLE CELLSPACING=3 BORDER=1 CELLPADDING=0>
 <TR>
    <TD><WEBOBJECT NAME=topTemplate></WEBOBJECT></TD>
 </TR>
 <TR>
    <TD><WEBOBJECT NAME=bottomTemplate></WEBOBJECT></TD>
 </TR>
</TABLE>
```

ReusableComponent.wod

```
topTemplate: RCComponentContent { templateName = "topTemplate"; }

bottomTemplate: RCComponentContent { templateName = "bottomTemplate"; }
```

MultipleComponentContent

Consumer

Parent.html

```
<WEBOBJECT NAME=myMCC_RC>
    <WEBOBJECT NAME=topTemplate>
        <Some random content>
    </WEBOBJECT>
    <WEBOBJECT NAME=bottomTemplate>
        <Some random content>
    </WEBOBJECT>
</WEBOBJECT>
```

Parent.wod

```
myMCC_RC: MCCRC { }

topTemplate: RCTemplate { templateName = "topTemplate"; }

bottomTemplate: RCTemplate { templateName = "bottomTemplate"; }
```

Demo

Multiple Component Content

Multiple Component Content

- Requires fewer supporting components
  - In the example: just the parent (consumer) and the RC are needed
  - Without MCC, a minimum of two extra components are needed
- Requires less complexity
  - No special protocol needed when creating child content for the Menu and Header
    - Passing state to RC to tell it what subcomponent and bindings to use
  - You can still use custom components in custom regions, but don’t have to

Streaming Requests/ Responses

WOFileUpload Streaming Options

- No streaming: the data binding
- streamToFilePath binding
- outputStream binding
- inputStream binding

streamToFilePath

- You provide a path, WO streams upload data to it
- Similar to outputStream: easier but less control
  - No stream object to create
  - Just set file name
- Overwrite binding available
- fileName binding set first so can use that info when providing streamToFilePath value

InputStream vs OutputStream

With outputStream:

- Data streamed for you during takeValues
- By time your action is invoked, data resides where outputStream pointed
- outputStream must exist by your construction
- You can have more than one WOFileUpload on the FORM in any order

InputStream vs OutputStream

With inputStream:

- Data doesn't start streaming until you assume control in your action method
- It's kind of like your action method participates in takeValues
- You read from inputStream provided by WO and then decide what to do with the bits
- WOFileUpload must be last element in FORM
- Only one WOFileUpload is allowed in a request
- Can't use multipleSubmit

InputStream vs. OutputStream

With inputStream:

- You have more control of the input data
- Maybe you don’t want to write to anything in full
- Maybe you want to stream to another server or DB
- Maybe you want transform stream data into something else

Streaming Responses

- Say you have a large file you want to download
- Create an InputStream based on it
- Assign it to WOResponse returned using setContentStream
- Return the WOResponse

Tips and Tricks

Component Initialization

Pages vs Reusable Components

- Both are WOComponents
- Initialization interactions different
- You create a page and control its initialization
- You don’t create Reusable Components directly

The Question

- When is it safe to initialize supporting structures?
  - For example: a list driving a popup
  - This might require hitting the DB or a complex calc
  - You don’t want to do it over and over
  - You may need binding values

One Solution

- Construct elements in their accessors lazily
- This is a pattern echoed throughout WOF & EOF

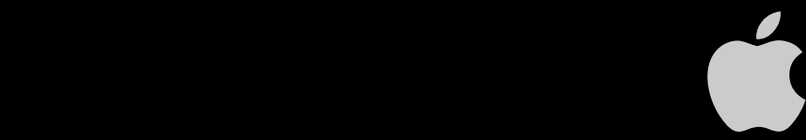
ReusableComponent.java

```
private NSArray listOfStuff;

public NSArray listOfStuff() {
    if( listOfStuff == null ) {
        // do fetch, pull bindings etc to assemble list
    }
    return listOfStuff;
}
```

Some Pitfalls

- Why not the constructor?
  - What if you need to change the list later?
  - What if you need state from a binding?
  - You might create objects unnecessarily
- Why not awake?
  - Bindings not available; not one-time call?
- Why not appendToResponse?
  - You might create objects unnecessarily; not one-time call



Another Pitfall

- Why not use the constructor combined with Session state instead of bindings?
  - Works around need to wait for bindings
  - Constructor only called once
  - I've got a more complex session
    - Less reusable: external coordination required
    - What if I never need to use the items constructed?
    - What if conditions change and I need a new list?

Advantages of Lazy Initialization

- Instead of you coordinating, things coordinate themselves
- Construction is deferred until bindings are available
- If the list is never needed, it's never created
- Easy cache invalidation
  - Changing agent just nulls the _ivar
  - If new list never needed, never constructed
  - Consumer of the list doesn't worry

Mysterious takeValues Failure

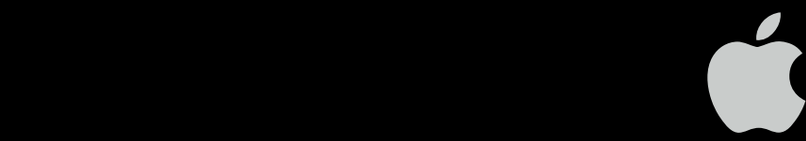
- The Problem:
  - Some but not all of a user's changes are missing...but only sometimes
  - On occasion exceptions are thrown for no apparent reason within takeValuesFromRequest()
  - Rearranging elements makes the problem go away

Mysterious takeValues Failure

- What's going on; an example:
  - A checkbox selection disables a view of a textfield
    - If checked, the textfield would be hidden
    - User input to the textfield is ignored
  - If checkbox is placed after the textfield, the layout driving state change happens after the textfield's values have been recovered
- Basic condition enabling problem:
  - User is allowed to submit changes that affect layout

Mysterious takeValues Failure
Solution

- Rearrange elements on page (brittle)
- Don’t let user change state that directly affects page layout (better)
  - Layout driving state changes should populate a holding variable
  - Holding variable only pushed into permanent storage in:
    - Action method
    - takeValuesFromRequest (post super)



Obtaining Values From Objects

- Typically used when iterating over and displaying multiple objects
- The beginner's urge is to use a "key" binding and call valueForKey() on the item to obtain a display string
- It's more flexible to have "item" and "displayString" bindings
  - Objects being displayed are not required to implement keyValueCoding
  - The parent is free to use custom methods to generate value

Not So Flexible:

Parent.wod

```
ListDisplay1:ListDisplay() {
    list = someArray;
    displayKey = “name”;
}
```

ListDisplay.html

```
<WEBOBJECT NAME=Repetition>
    <WEBOBJECT NAME=Label></WEBOBJECT>
</WEBOBJECT NAME=Repetition>
```

ListDisplay.java

```
public NSArray someArray;
public EOEnterpriseObject item;

public String displayString() {
    return
item.valueForKey(displayKey());
}
```

ListDisplay.wod

```
Repetition:WORepetition {
    list = someArray;
    item = item;
}

Label:WOString {
    value = displayString;
}
```

More Flexible:

Parent.wod

```
ListDisplay1:ListDisplay() {
    list = someArray;
    item = item;
    displayString = displayString;
}
```

ListDisplay.html

```
<WEBOBJECT NAME=Repetition>
    <WEBOBJECT NAME=Label></WEBOBJECT>
</WEBOBJECT NAME=Repetition>
```

Parent.java

```
public NSArray someArray;
public MyClass item;

public String displayString() {
    return “[” + item.name() + “]”;
}
```

ListDisplay.wod

```
Repetition:WORepetition {
    list = someArray;
    item = ^item;
}

Label:WOString {
    value = ^displayString;
}
```

WOComponentContent in tables

- Most table components repeat both rows and columns inside the component
- It's much more flexible to repeat only the rows inside the component and use WOComponentContent for the columns
- This allows parent to do *anything* for the table cells

Not So Flexible:

Parent.wod

```
Table:CustomTable {
    list = someArray;
    headers = “Item,QTY,Price”;
    displayKeys = “name,quantity,price”;
    displayAlign = “left,center,right”;
    displayStyle = “bold,,”;
    ...
}
```

Table.html

```
<TABLE>
    <WEBOBJECT NAME=RowRepetition>
        <TR>
        <WEBOBJECT NAME=ColumnRepetition>
            <TD><WEBOBJECT NAME=Label></WEBOBJECT></TD>
        </WEBOBJECT NAME=ColumnRepetition>
        </TR>
    </WEBOBJECT NAME=RowRepetition>
</TABLE>
```

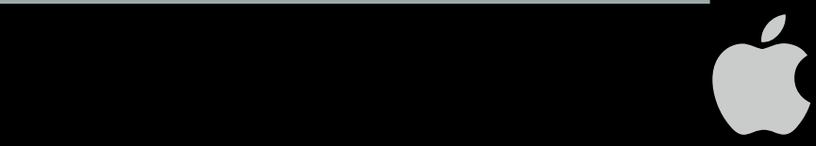
More Flexible:

Parent.html

```
<WEBOBJECT NAME=CustomTable>
    <TD><B><WEBOBJECT NAME=ItemString></WEBOBJECT></B></TD>
    <TD ALIGN=CENTER><WEBOBJECT NAME=QuantityString></WEBOBJECT></TD>
    <TD ALIGN=RIGHT><WEBOBJECT NAME=PriceString></WEBOBJECT></TD>
</WEBOBJECT NAME=CustomTable>
```

Parent.wod

Table.html



Dr. Evil

Why you really came to WWDC

Parent Talking Directly to Child

- The parent can send messages directly to the reusable component
- The value is not available until the child starts generating its response
- This can be used to allow sibling components to communicate

ReusableComponent.java

```
public void appendToResponse(WOResponse response, WOContext context) {
    setValueForBinding(this, “subcomponent”);
    super.appendToResponse(response, context);
}
```

Array Bindings

- Makes it much more convenient for parents to supply arrays of strings to bindings that need it
- Otherwise, you end up writing little bits of tedious code to create arrays

Array Bindings

Parent.wod

```
Reusable1:ReusableComponent {
    binding1 = “left,right,up,down”;
    binding2 = someArray;
}
```

ReusableComponent.java

```
public NSArray binding1() {
    return arrayBindingNamed(“binding1”);
}

public NSArray arrayBindingNamed(String name) {
    value = valueForBinding(name);
    if ( value instanceof NSArray ) {
        return (NSArray)value;
    } else if ( value instanceof String ) {
        return NSArray.componentsSeparatedByString((String)value, “,”);
    } else {
        throw new RuntimeException(“Error!”);
    }
}
```

Smart Bindings

- Problem: Lots of elements needing minor code intervention
  - Prepend same path to dozens of WOImage src bindings
  - Drive dozens of WOString values with same method but each taking different arguments
- Solutions include
  - Creating a little component that performs the intervention
  - Creating dozens of trivial methods
  - Use a “Smart Binding”

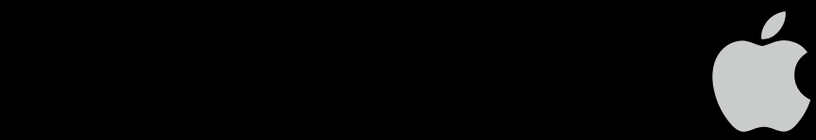
Smart Bindings

- Example: Suppose you need to convert amounts based on native currency?
  - Without Smart Bindings

```
value = orderDetalSubtotal;
value = selectedOrderTotal;
value = selectedOrderShippingCost;
```

  - With Smart Bindings

```
value = nativeCurrency@aOrderDetal.subtotal;
value = nativeCurrency@selectedOrder.total;
value = nativeCurrency@selectedOrder.shippingCost;
```



Smart Bindings How?

- Override WOComponent's valueForKeyPath

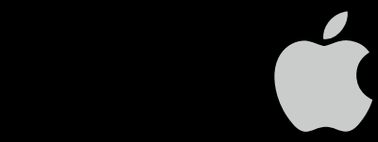
Parent.java

```
// Assume methods argFromKeyPath and nativeCurrency exist

public Object valueForKeyPath(String keyPath) {
    if ( keyPath != null ) {
        if ( keyPath.indexOf("nativeCurrency@") == 0 ) {
            String argValue = argFromKeyPath(keyPath);
            Object val = super.valueForKeyPath(argValue);
            return nativeCurrency(val);
        }
    }
    return super.valueForKeyPath(keyPath);
}
```

Using Your Own Components

- WOComponents merely know how to display themselves
- You can develop your own Components that “subclass” WOComponents
- WOString -> MYString
- WORepetition -> MYRepetition
- WOImage -> MYImage
- Etc.



Using Your Own Components

- You can make components that
  - Know how to edit themselves
  - Write themselves to disk as HTML
  - Archive themselves as XML
  - Add/Remove other components from the component tree

Examples

Project Wonder

http://wonder.sourceforge.net/

Q & A

http://developer.apple.com/wwdc2003/urls.html

Roadmap

616 WebObjects EOF— Advanced Topcics	Russian Hill Wed., 5:00pm
FF007 WebObjects Feedback	WebObjects Thur., 9:00am
624 Webobjects— Creating Web Services	Nob Hill Fri., 3:30pm
603 WebObjects Technical Overview	Russian Hill Tues., 10:30am

Who to Contact

Andreas Wendker
Manager, WebObjects Engineering
webobjects@apple.com

Bob Fraser
Senior Product Manager, WebObjects and Java
webobjects@apple.com

Katherine Wenc
Manager, Technical Publications
webobjects@apple.com

Enterprise-Level WebObjects Support And Consulting
AppleCare Enterprise Services
services@apple.com
(800) 630 2775

http://developer.apple.com/wwdc2003/urls.html

Reference Library

Resources

Product Information

http://www.apple.com/webobjects

Developer Support

http://developer.apple.com/webobjects

Enterprise-Level WebObjects Support And Consulting

http://www.apple.com/services/technicalsupport

http://www.apple.com/services/consulting

Mailing List

WebObjects Announcements

webobjects-announce@apple.com

WWDC 2003

Worldwide Developers Conference